Implementing Direct Anonymous Attestation on TPM 2.0

Noah Robert Luther

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science and Applications

Jung-Min "Jerry" Park, Chair

Ali Butt, Co-Chair

Gang Wang

May 9, 2017

Blacksburg, Virginia

Keywords: Direct Anonymous Attestation, TPM, Trusted Computing

Implementing Direct Anonymous Attestation on TPM 2.0

Noah Robert Luther

(ABSTRACT)

Numerous organizations have pressed in the past several years for improved security and privacy in online interactions. Stakeholders have encouraged the adoption of privacy-enhancing technologies, utilization of microcontrollers and hardware devices for key storage and attestation, and improvements to the methods and policies used for authentication. Cryptographers and security engineers have responded to these calls. There have been numerous papers published in the last decade on topics such as private information retrieval and anonymous authentication and the Trusted Computing Group (TCG) has released a version 2.0 standard for Trusted Platform Modules (TPM). Adoption and implementation of these techniques, however, has been lacking. Although the TPM 2.0 specification was released in 2014 there are no reference implementations of direct anonymous attestation algorithms compatible with the hardware. The purpose of this work is to implement and discuss the implementation of direct anonymous attestation on TPM 2.0 and to consider the scalability and performance of direct anonymous attestation schemes operating on real-world TPM devices.

Implementing Direct Anonymous Attestation on TPM 2.0

Noah Robert Luther

(GENERAL AUDIENCE ABSTRACT)

Numerous organizations have pressed in the past several years for improved security and privacy in online interactions. Stakeholders have encouraged the adoption of new technologies for authentication to reduce the instances of fraud and identity theft. Researchers and engineers have developed standards and devices that aim to simultaneously improve security while maintaining user privacy. In particular, an organization called the Trusted Computing Group has released standards for a device called a Trusted Platform Module. This device is built in to many modern personal computers and is designed to allow users to authenticate without compromising their privacy. Even though the version 2.0 standard was released in 2014, however, there are no reference implementations of standardized privacy-preserving authentication algorithms compatible with the device. The purpose of this work is to implement algorithms for authentication utilizing a Trusted Platform Module and to discuss their performance in the real world.

Acknowledgments

I thank my advisor Professor Jerry Park of the Bradley Department of Electrical and Computer Engineering at Virginia Tech for his support, patience and guidance. I thank the members of my research group in Prof. Park's ARIAS lab for their support and collaboration, particularly Pranav, Vireshwar, and He. I thank my colleagues at Qualcomm and MIT Lincoln Laboratory for welcoming me and giving me the opportunity to work with them for the past several summers, particular Joseph, Bren, and Prasanth from Qualcomm and Emily, Sophia, Arkady, Rob, Neal, Robert, and Cem from Lincoln. I thank the past and present members of the Cyber Security Club at Virginia Tech for giving me responsibility, camaraderie, and endless opportunities to learn. I thank the IT Security Office at Virginia Tech for continuing to welcome me to distract and learn from them long after I stopped working there. I thank my friends in Blacksburg, San Diego, Boston, and elsewhere. Finally, I thank my parents, my brothers, and my family for their love and belief in me. You have all enriched my life. Thank you.

Contents

List of Figures

Chapter 1

Introduction

According to Internet Association estimates the 'Internet sector' contributes roughly six percent of the United States gross domestic product per year, trending upwards. As online transactions are a large portion of the American and global economies, fraud and identity theft have become major problems. The National Strategy for Trusted Identities in Cyberspace (NSTIC) [31] was released by the Obama administration in April 2011. The document expressed the United States government's concern that online transactions were subject to fraud and identity theft in greater numbers as well as raising concerns over privacy protections for consumers. The NSTIC proposed the development and application of privacy-enhancing technologies such as privacy-preserving authentication as a way to reduce the risks posed by these crimes while simultaneously protecting the privacy of consumers.

Privacy-enhancing technologies are techniques, products or systems that enable the use of

technology while preserving the privacy of users under certain assumptions. Some examples include anonymous routing with The Onion Router [25], secure and deniable communications with Off The Record (OTR) messaging [9], and private information retrieval [24]. Privacy-preserving authentication is a particular privacy-enhancing technology that simultaneously provides authentication akin to traditional digital signature schemes while allowing the signer to unlink their identity from the signature. In this thesis we will discuss a particular type of privacy-preserving authentication called direct anonymous attestation (DAA) [10], in which the key material used to authenticate a particular computing platform is stored on a hardware module called a trusted platform module (TPM) [46, 45].

Trusted platform modules have been shipped with many consumer computing devices in the past decade. They can be used to generate cryptographic key material as well as produce digital signatures and attest to the health of the devices [45]. Attestation is used by a device to prove to a verifier that the device or some installed software meets some requirement [3]. Direct anonymous attestation provides this proof while simultaneously allowing the device owner to maintain their privacy [10]. The NSTIC and Industry stakeholders in the Trusted Computing Group (TCG) identify direct anonymous attestation and the TPM as valuable elements in the push to improve online authentication while maintaining user privacy [31, 10].

1.1 Purpose and Goals

In this thesis we seek to implement and evaluate direct anonymous attestation on a hardware TPM version 2.0. We investigate the performance degradation of a few direct anonymous attestation schemes as the number of revoked signatures grows. We discuss the practical benefits of a slightly relaxed security model for direct anonymous attestation and the resulting impact on real-world performance of these privacy-preserving authentication schemes.

1.2 Challenges

1. Reading and understanding the TPM specification. It was designed to be easily maintainable and is correspondingly challenging to read [45, 3]. We discuss strategies to navigate the specification in a later chapter.

2. Configuration of TPM commands and other software libraries for interoperability.

3. Determining what changes will be necessary as a result of weaknesses in the TPM 2.0 standard [49, 38, 16]. One of the TPM commands that is used for DAA utilizes the TPM as a Static Diffie-Hellman oracle which lowers the security level of this implementation.

1.3 Contributions

1. Software implementations of two direct anonymous attestation schemes utilizing a hardware TPM version 2.0: EPID, a variant of an Intel standardized DAA scheme, and LASER, which can provide either full DAA or a version with relaxed unlinkability requirements that aims to improve real-world performance.

2. Performance analysis and comparison of EPID & LASER.

To the best of our knowledge these are the first implementations of DAA schemes utilizing a hardware TPM 2.0 device.

1.4 Structure of Thesis

This thesis is organized by first discussing background material including privacy-preserving authentication, the trusted platform module (TPM) standards, and direct anonymous attestation (DAA) in Chapter 2. A survey of related work is provided in Chapter 3. Supporting libraries, documentation and implementation are discussed in Chapter 4. The results are evaluated in Chapter 5. Finally, conclusions and opportunities for future work are provided in Chapter 6.

Chapter 2

Background

In this chapter we will discuss various types of privacy-preserving authentication. We will then discuss the TPM standard and TPM hardware. Finally, we will discuss the key ideas of direct anonymous attestation and some of the problems that make it challenging to implement an efficient protocol.

2.1 Privacy-preserving Authentication

Digital signatures can be used to authenticate messages and entities but traditional signature schemes require that the individual that signs the message be uniquely identifiable. Privacy-preserving authentication is a cryptographic technique that allows authentication but does not require that data be attributed to a particular source. Some examples of privacy-preserving authentication schemes are ring signatures [43] and group signatures [21].

Both types of signature schemes have variants based on RSA or elliptic curve cryptography.

2.1.1 Ring Signatures

Ring signatures allow signers to sign a message as a member of an arbitrary ring of possible signers. The signer can select the members of the ring even without the permission of the other signers. They allow total flexibility on the part of the signer, cannot be linked back to the owner cryptographically, and cannot be revoked [43]. They were developed by Ron Rivest et. al. and first published in a paper titled 'How To Leak a Secret' [43]. These signatures are produced by utilizing the public keys of all members of the ring including the actual signer and both the private key of the actual signer. Although these signatures provide numerous benefits to signers, they have less applications than other types of privacy-preserving signature schemes because they do not allow for revocation or for group management by a third-party service provider.

2.1.2 Group Signatures

Group signatures, on the other hand, were designed to allow a third-party group manager to determine the makeup of the group as opposed to the actual signers that perform this function in ring signatures [21]. Any member of the group can produce a valid and verifiable signature proving that they are a member of the group. Often, these signatures can be 'opened' by the group manager to determine which of the group members actually produced

the message if the manager believes the member is acting out [21]. This implies that the anonymity guarantee provided by a group signature relative to a traditional digital signature is weaker than would be provided by a ring signature. Some signature schemes similar to a group signature do not have the 'open', property, however, including direct anonymous attestation (DAA) [10]. These schemes do not allow the group issuer to deanonymize signers but may allow the group manager to add new members to the group or to revoke membership.

2.2 Pairing-based Cryptography

As mentioned previously, both ring and group signatures have variants based on RSA and based on elliptic curve cryptography. Many modern group signature schemes are based on bilinear pairings over elliptic curves rather than RSA. Pairing-based schemes allow for more efficient implementations of many cryptographic systems. As the schemes implemented in this thesis constitute privacy-preserving authentication schemes utilizing pairing-based cryptography, it is appropriate at this time to discuss cryptographic constructions based on pairings.

2.2.1 Bilinear Pairings

Suppose G_1 and G_2 are two groups of prime order q and G_T is a group. A bilinear pairing is a special kind of map $e : G_1 \times G_2 \rightarrow G_T$ satisfying the bilinearity and non-degeneracy properties [26].

(1) Bilinearity requires that for every $P \in G_1, Q \in G_2, a, b \in \mathbb{Z}_q^*$, it holds that $e(P^a, Q^b) = e(P, Q)^{ab}$.

(2) Non-degeneracy implies that given g_1 and g_2 are the generators G_1 and G_2, respectively, then $e(g_1, g_2) \neq 1$.

Typically, G_1 and G_2 are elliptic curve groups and G_T is a finite field. Some examples of pairings that meet these criteria are the Weil and Tate pairings [26, 5]. Several direct anonymous attestation schemes and other privacy-preserving authentication schemes utilize cryptographic constructions based on bilinear pairings [19, 11, 17, 8, 7, 12, 14, 15, 18, 26].

2.2.2 Zero-Knowledge Proofs of Knowledge

Proof of knowledge protocols allow one to prove knowledge of discrete logarithms and validity of relations between discrete logarithms without revealing any information about them [27, 19]. These proofs are interactive; they require multiple rounds of communication between the prover and the verifier. In the random oracle model, a signature scheme can be designed that accomplishes the same goal without interactivity by utilizing the Fiat-Shamir hueristic [27]. Many privacy-preserving authentication schemes utilize zero-knowledge proofs or signatures of knowledge to prove that a platform or user possesses a valid credential without exposing that credential to the verifier.

2.2.3 BBS+ Signatures

The direct anonymous attestation schemes discussed and implemented in this thesis both utilize BBS+ signatures. The BBS+ signature scheme is a signature scheme based on bilinear pairings that allows more efficient generation of signatures than other constructions. Suppose G_1, G_2 are groups of prime order p and G_T is a group such that $e : G_1 \times G_2 \to G_T$ is a bilinear pairing.

(1) Setup: Select g_1, h_1, h_2, h_3 uniformly at random from G_1, g_2 uniformly at random from G_2, and γ uniformly at random from $\mathbb{Z}_p^*$. Compute $\omega = g_2^{\gamma}$. The public key $pk = (g_1, h_1, h_2, h_3, g_2, \omega)$ and the private key is γ.

(2) Sign: Input pk, γ, and two messages $f, x \in \mathbb{Z}_p^*$. Select y, z uniformly at random from $\mathbb{Z}_p^*$ and compute $A = \left(g_1 \cdot h_1^f \cdot h_2^x \cdot h_3^y\right)^{1/(\gamma+z)}$. The signature is $\sigma = (A, y, z)$.

(3) Verify: Input pk, a signature σ, and two messages f and x. Verify with the bilinear pairing operation that $e(A, \omega \cdot g_2^z) = e(g_1 \cdot h_1^f \cdot h_2^x \cdot h_3^y, g_2)$.

The verification holds since $\omega = g_2^{\gamma}$, so $\omega \cdot g_2^z = g_2^{\gamma+z}$, and the pairing operation thus yields that $e(A, \omega \cdot g_2^z) = e(g_1 \cdot h_1^f \cdot h_2^x \cdot h_3^y, g_2)^{(\gamma+z)\cdot 1/(\gamma+z)}$. Note that this verification works even though the verifier never knows the private key γ.

2.3 Trusted Platform Modules

Trusted platform modules (TPM) are hardware devices typically shipped as an internal component of a consumer or business computer. They are dedicated cryptoprocessors that can be used to generate cryptographic keys with an internal random number generator as well as to provide device attestations. Trusted platform modules were designed by the Trusted Computing Group (TCG), an industry standardization body with the goal of developing industry standards for trusted computing hardware and software that can be used for secure data storage and online transactions while maintaining privacy for users [3].

2.3.1 TPM Standards

The first TPM standard was TPM 1.1b [47]. It was released in 2003 and was capable of RSA key generation, storage, authorization, and device-health attestation [3]. This first standard was non-specific, and different hardware vendors produced hardware-incompatible devices based on the standard. The TPM 1.2 standard [46] was developed between 2005 and 2009 and standardized the software interface. The TPM 1.2 standard also introduced direct anonymous attestation, which will be discussed in greater detail in the next section [10, 46]. The most recent standard, TPM 2.0, introduced a great number of changes from the TPM 1.2 specification. The TPM 2.0 standard introduced flexibility in algorithm design; prior to TPM 2.0 the standard required SHA-1 as the only hash algorithm [45]. It also introduced symmetric encryption and the option to utilize elliptic curves as an alternative to RSA [45].

This thesis is concerned with the TPM 2.0 standard; any references to TPM devices refer to TPM 2.0. ISO standards for Trusted Platform Modules can be found in the references [32, 35].

Both initially and throughout the development of TPM standards, the business requirements of stakeholders required that the hardware production cost be very cheap. This requirement led to the design of a chip with minimalist hardware functionality augmented by software. Consequently, the specification of the TPM standards started and remains very complex [47, 46, 45, 3]. In fact, the TPM 2.0 standard is split into four parts and totals more than 1400 pages. This complexity is likely part of the reason that TPM devices are underutilized in practice, the software interface has historically been challenging to learn!

2.3.2 Device Attestation

Device attestation is a procedure by which a device can present evidence about the state of the device or the software that it is currently running [3]. One example of an attestation service is device health attestation in Windows 10 enterprises that allows hardware monitoring of security settings [41], but there are numerous potential applications. Many attestation applications require that the device submitting a report be authenticated. This could be implemented using traditional digital signatures and certificates but such a naive implementation leads to several problems. First, if every hardware device has a copy of the same key, there is no way to revoke a device for misbehaving [10, 3]. If every hardware device has a

unique key mapped to the owner's identity, then privacy is nonexistent [10].

The Trusted Computing Group proposed an early solution to this problem called a privacy certificate authority (privacy CA) in the TPM 1.1b standard, but this solution had a number of problems of its own [47, 10, 3]. The privacy CA serves as a trusted third party that knows the unique identifiers of every TPM device. When a TPM device wishes to attest to a verifier, it generates a key, signs it with its unique identifier, and then has the privacy CA issue a certificate for the new key. Unfortunately, this scheme both requires the privacy CA to know the identity of the TPM owner and requires that the privacy CA be highly available, as it must interact with the platform for every attestation.

In order to solve the problems associated with the privacy CA, the TCG developed a new algorithm called direct anonymous attestation for inclusion in the TPM 1.2 standard [10, 46].

2.4 Direct Anonymous Attestation

Direct Anonymous Attestation (DAA) was first proposed by Brickell et. al. in 2004 [10]. DAA is a cryptographic protocol that enables attestation of a computing platform without compromising the platform-owner's privacy. It achieves this by decoupling the signatures themselves from the identity of the platform user. A typical DAA scheme involves three entities; platform, issuer, and verifier. The issuer generates and issues credentials to platforms and revokes compromised platforms. Each platform in a direct anonymous attestation scheme is comprised of a host computer and a TPM. An unrevoked platform is able to gen-

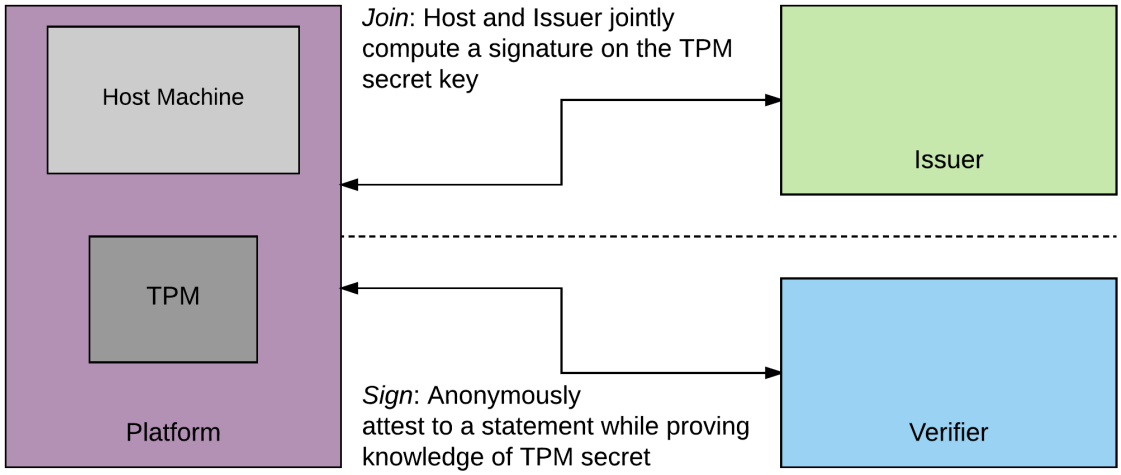

Figure 2.1: Direct Anonymous Attestation

erate signatures that prove to a verifier that the platform that produced the signature was indeed issued a credential by the issuer. The verifier validates these proofs but does not learn which platform produced the signature by doing so. A visual representation of the basic idea of direct anonymous attestation can be seen in Figure 2.1. Direct anonymous attestation has been standardized by ISO; there is an existing standard and an updated standard under development [34, 33].

2.4.1 Signature-Based Revocation

As the issuer and the verifier do not know who produced a signature, the only way to revoke a misbehaving platform in a DAA scheme without knowing the platform's private key is to do signature-based revocation [22, 37, 13, 15]. This feature did not exist in the original DAA specification [10] but was introduced in Enhanced Privacy ID (EPID) [13, 15] Every signature produced by a platform utilizing DAA schemes that support signature-based revocation has

a basename and another group element that is the basename raised to the TPM secret. To prove that the platform did not produce any of the revoked signatures, the platform must demonstrate with a zero-knowledge proof for each revoked signature that its TPM secret could not have been used to produce this pair [15, 37]. This cost grows linearly with the number of revoked signatures, and thus constitutes a large portion of the overhead of a DAA scheme in practice. Many of the performance issues that DAA faces are rooted in this problem.

2.4.2 Anonymity and Unlinkability

Anonymity and unlinkability are two related but different concepts in the workspace of DAA. Anonymity is concerned with disconnecting the identity of an individual from the signatures they produce. Unlinkability is concerned with making it impossible for two signatures produced by the same entity to be recognizable as such [10]. It is possible for a privacy-preserving authentication scheme to maintain the anonymity of an entity without providing unlinkability or while providing only partial unlinkability. The converse is not true; if a scheme does not provide anonymity guarantees, then clearly two messages signed by an identified individual would be linkable.

Chapter 3

Related Work

There is a sizeable body of literature involving the design of direct anonymous attestation schemes based on various assumptions. Fewer papers have been published on the actual implementation of these schemes. In this chapter we will investigate the published works on the topic of implementing direct attestation either in software or by utilizing physical TPM devices. We will also discuss discovered problems in the TPM specification and ongoing efforts to repair weaknesses that reduce the security level of DAA implementations.

3.1 Direct Anonymous Attestation Schemes

Direct anonymous attestation based on RSA was first proposed by Brickell et. al. in 2004 [10]. The cryptographic primitive was designed to allow remote authentication without uniquely identifying the user of the platform being authenticated. Since this scheme was

originally proposed a number of papers have been published developing new DAA schemes and improvements to existing schemes. In particular more recent papers have moved away from RSA-based DAA to develop schemes based on bilinear pairings and elliptic curves.

3.1.1 Enhanced Privacy ID (EPID)

Enhanced Privacy ID (EPID) is a direct anonymous attestation scheme developed by Intel that serves as a direct successor to the original DAA scheme. It moves away from RSA-based public key algorithms to incorporate bilinear pairings and elliptic curve cryptography [13]. It also introduces the notion of signature-based revocation to allow revocation of misbehaving platforms even when the private keys generated by the platform TPM are not exposed [13]. This means that EPID utilizes two separate revocation lists; one for the platforms with exposed keys and one containing all signatures that have previously been revoked. Since EPID is the basis for many direct anonymous attestation schemes developed in the subsequent years and is one of the algorithms implemented in this thesis, we will briefly discuss the specification now.

EPID Specification

The specification includes five algorithms; `Setup`, `Join`, `Sign`, `Verify`, and `Revoke` [13]. The `Setup` algorithm is used by the issuer to generate a group public key `gpk` and an issuer private key `isk`. The `Join` algorithm allows platforms to request entry into the group. If the

platform is permitted, the issuer and platform jointly compute a BBS+ signature on the key of the platform without the issuer learning this key. The issuer utilizes `isk` to produce this signature. The BBS+ signature serves as the platform's secret key `sk`. The `Sign` algorithm has a platform utilize its own secret key `sk` and the group public key `gpk` to compute proofs of membership and nonrevocation for the verifier. The proof of membership demonstrates to the verifier in zero-knowledge that the platform successfully produced a BBS+ signature on its secret in concert with the issuer. The proofs of nonrevocation prove to the verifier, for each revoked signature, that the platform could not have possibly produced that signature. `Verify` simply accepts these proofs and validates that they are correct. `Revoke` either adds an exposed private key to the list of exposed private keys or adds a signature to the list of revoked signatures.

3.1.2 Techniques to Improve Revocation

A number of research efforts in the area of privacy-preserving authentication have focused on attempts to make revocation more efficient in practice. As discussed previously, EPID's signature-based revocation leads to a dramatic increase in the cost of producing a signature as the number of revoked signatures grows. For every additional revoked signature, it becomes necessary for the signer to produce another proof of nonrevocation by computing a zero-knowledge proof [13]. The base computational cost of a proof of membership is not large but the cost of proving nonrevocation in practical settings quickly becomes massive. Indeed, studies of group signatures and anonymous credential schemes have shown that revocation

remains the major bottleneck in their performance [40].

Group Resetting

The authors of EPID propose group resetting as a viable means to prevent performance degradation due to long revocation lists [13]. This means that the issuer would require all platforms to obtain new membership credentials when the revocation list grew too long, and would not issue credentials to platforms that had already been revoked. Unfortunately there is little information available on how this resetting should work in practice. Since group resetting would require every platform to rejoin the group, it could represent a large computational cost itself. Without knowing a specific application of DAA including the size of the network and the rate of revocation, it is hard to simulate a group resetting scheme in a reasonable way. There is no reason that modifications to existing schemes that improved DAA in the presence of large revocation lists must be incompatible with group resetting schemes, however.

Probabilistic Revocation

One attempted solution to this problem is to make the revocation check procedure probabilistic, allowing false positives (unrevoked signatures appear revoked) but not false negatives (revoked signatures are guaranteed to appear revoked). In Group Signatures with Probabilistic Revocation (GSPR) [37] the verifier does not have to perform an iterative check of each revoked item in the revocation list to validate a given signature. Instead, the issuer maps a

vector of +1s and -1s called an 'alias code' to each revocation list entry and distributes this map to the verifiers. As revocation list entries are a fixed portion of all candidate signatures, verifiers can then map this portion of a candidate signature to an alias code and check if the code is included in the revocation list. This allows the verifier to determine probabilistically whether or not the signer is revoked without a linear increase in the cost of signature production.

Cryptographic Accumulators

Cryptographic accumulators are one-way membership functions; they allow queries as to whether or not a particular item is a member of a set without revealing the members of the set [6]. Better yet, the set is represented with a short single value that can be communicated with low cost. Dynamic accumulators also offer the option to add or remove elements from the set and recompute the accumulator value. Accumulators typically allow efficient generation of 'witnesses' to prove that an item is in the set, certain more advanced accumulators also offer proofs of non-membership. Until recently when a dynamic accumulator was updated and elements were added or removed, these witness values needed to be recomputed by all of the verifiers.

The most recent dynamic accumulators show promise for privacy-preserving revocation applications as they provide a means to keep track of revoked entities without dramatic increases to the computation cost. Braavos is a dynamic accumulator that only requires witness updates when elements are removed from the accumulator, but not when new items are added

[4]. Given the accumulator is keeping track of revocation list entries, this means that new additions to the revocation list do not invalidate verifiers' stored witnesses associated with the entries that had already been in the list.

Online/Offline Tradeoff

Rather than trying to reduce the overall complexity of the revocation check, another alternative is to consider splitting the computation cost into an offline phase and an online phase. If there is substantial opportunity for precomputation, then an expensive offline phase might be reasonable given that the online phase has low cost. One major issue facing EPID is that its computational cost associated with proofs of nonrevocation appears in the `Sign` algorithm and thus must be recomputed online whenever a message is signed. If the most expensive component of the algorithm was obtaining credentials instead of signing messages, the online cost of the scheme could be greatly reduced. This means that the delay in the latency-dependent phase of the scheme can be minimized. One example of a direct anonymous attestation scheme that utilizes an online/offline tradeoff will be discussed in the next section.

3.1.3 Lightweight Anonymous Attestation Scheme with Efficient Revocation

Lightweight Anonymous Attestation Scheme with Efficient Revocation (LASER) is a DAA scheme which exhibits an online/offline tradeoff in order to improve practical performance in the presence of large revocation lists [36]. The scheme moves most of the heavy computation to an offline phase during which a platform obtains credentials and has a constant cost for producing a signature once credentials have been obtained. It also introduces another notion called 'adaptable unlinkability' which, at the option of the platform operator, reduces the overall complexity of the computation by relaxing the unlinkability requirement. LASER is one of the two direct anonymous attestation schemes implemented on TPM 2.0 in this thesis. We will next discuss the specification of LASER followed by considering the benefits of adaptable unlinkability.

LASER Specification

Similar to EPID, LASER has `Setup`, `Join`, `Sign`, `Verify`, and `Revoke` algorithms. It also adds a few additional algorithms called `GetSignCre` and `SelectAliasCre`. The `Setup` algorithm is used by the issuer to produce the group public key `gpk` and an issuer secret key `isk`, similarly to EPID. The `Join` algorithm in LASER is used to obtain a 'membership credential' which is a BBS+ signature on a platform secret key produced jointly by the issuer and the platform. Again, this part of the specification is quite similar to EPID.

The `GetSignCre` operation is a major divergence. In `GetSignCre`, the platform obtains an additional 'signing credential' at a lower level in a hierarchy than the membership credential. In order to obtain a new signing credential, the platform must prove that it has not been revoked by iteratively proving it could not have produced any of the revoked signatures. This operation, however, is no longer required in the `Sign` algorithm, which becomes constant time. `GetSignCre` also allows the platform to generate a number of 'alias tokens' associated with each sign credential. The use of different credentials to `Sign` has an impact on the unlinkability guarantees of the scheme. This will be discussed in detail in the next section. The decision making process for a platform is encoded in the `SelectAliasCre` algorithm. `Sign` always requires an alias token to be used to produce a signature. The `Verify` and `Revoke` operations are similar to DAA and EPID with the difference that the verifier must only validate that the signer is a member of the group which takes constant time.

Adaptable Unlinkability

EPID requires that all signatures be fully unlinkable from each other. This means that there is no way for a verifier or issuer to tell from the signatures themselves that they were produced by the same platform [13]. This same feature can be provided by LASER, but the scheme performs better when this requirement is not needed by the platform. Figure 3.1 on the following page shows how using or reusing various credentials produced in LASER provides different unlinkability guarantees.

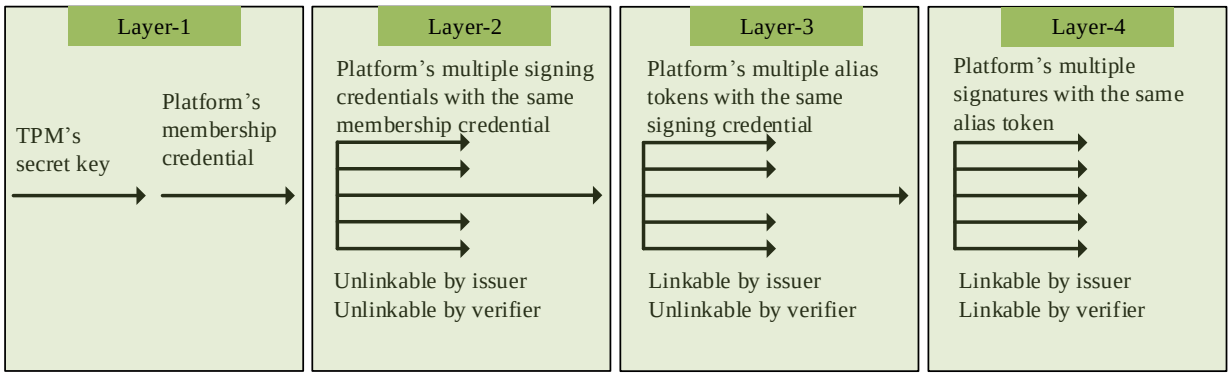

Figure 3.1: Adaptable Unlinkablility in LASER.

In brief, signatures produced using alias tokens from the same signing credential can be linked by the issuer, signatures produced by the same alias token are fully linkable by both the issuer and the verifier, and signatures produced under different signing credentials cannot be linked by the verifier or the issuer. The benefit of adaptable unlinkability is that in situations where the platform does not require that its transactions are unlinkable by the issuer they can utilize the same signing credential with multiple alias tokens. The expensive part of the algorithm is obtaining new signing credentials since it is at this moment that the platform must produce proofs of nonrevocation. By making it possible for the platform to choose to reuse signing credentials the cost of performing multiple signatures can be greatly reduced. This section is included in related works because the development of the LASER scheme was the work of another student. However, the author did contribute substantially to the development of the notion of adaptable unlinkability.

3.2 DAA Implementations

In this section we will discuss existing work on the topic of implementing direct anonymous attestation. We will first consider the implementation of DAA schemes in software and utilizing alternative hardware such as ARM TrustZone [2]. Next we will consider an existing analysis and comparison of a software-only and a TPM 1.2 implementation of DAA. Finally, we will discuss papers that consider how to translate a direct anonymous attestation scheme into an implementation on TPM 2.0 and address weaknesses in the TPM 2.0 specification but that stop short of providing a full implementation.

3.2.1 Software Implementations

Although implementations of direct anonymous attestation utilizing TPM 2.0 are not prevalent, there are several examples of DAA implementations utilizing software and technologies such as ARM TrustZone [50, 51, 48]. ARM TrustZone is a hardware-based security technology that separates trusted and untrusted operations between two 'worlds' [2]. Software in the non-secure world cannot access resources in the secure world without switching through a monitor [2]. ARM TrustZone and hardware TPM devices are not identical technologies, although they share certain similarities in design goals.

Some of these papers involve work on making DAA more feasible for mobile applications [50, 51], particularly when there is no need for signature-based revocation. Others compare multiple direct anonymous attestation schemes implemented on ARM TrustZone and utiliz-

ing various elliptic curve groups [48]. These benchmarks are useful for their own sake, but do not give good estimates of the time taken by a hardware TPM device to compute the same schemes. The majority of papers on direct anonymous attestation accompanied by an implementation utilize pure software or hardware support from a technology like ARM TrustZone rather than an actual TPM device.

One recent implementation of a direct anonymous attestation scheme in software deserves particular interest. Intel has published a software development kit for EPID and the third release was published in late November of 2016 [1]. This software package allows those interested in developing applications that utilize EPID to build the algorithm into their application entirely in software. However, this implementation does not utilize a hardware TPM.

3.2.2 TPM 1.2 Implementation

Prior to the proliferation of the TPM 2.0 standard, one research group did implement direct anonymous attestation on TPM 1.2 [20]. They utilized OpenSSL [42] and an Infineon TPM 1.2 for their TPM implementation and also produced a software only implementation of DAA. Both implementations were built into a TLS application. Since they were utilizing TPM 1.2, their DAA implementation used RSA-based DAA and was not concerned with revocation. The results of their implementation were clear; the TPM implementation of DAA was substantially slower than a software-only implementation and both increased the

cost of a TLS connection [20]. As this implementation utilized TPM 1.2, was tested on a different platform, and was integrated into a particular application, their specific results were not comparable to those obtained in this thesis. Even so, their conclusion that it may be beneficial to move more of the computational cost of a DAA scheme to the server side and to reduce the computation performed by the TPM to the minimum are worth noting.

3.3 Discussions of TPM 2.0 APIs for DAA

The TPM 2.0 interface was designed to be able to support a number of different cryptographic schemes with a single TPM signature primitive due to hardware design constraints [23]. The design allows for the implementation of DAA, psuedonym systems, and conventional digital signatures with low hardware complexity. Low hardware complexity often results in more complex supporting software since the interface needs to be flexible with many options.

The TPM 2.0 DAA interface works in exactly this way. With a handful of TPM commands including `TPM2_Create`, `TPM2_CreatePrimary`, `TPM2_Load`, `TPM2_Commit`, `TPM2_Sign`, `TPM2_Hash`, and possibly a few supporting commands, it is possible to implement all of the TPM functionality of a DAA scheme. The paper by Chen and Li outlining the API and how it ought to be used to implement DAA was published in CCS 2013 [23]. Unfortunately, later research demonstrated that there were problems with this API that result in weakened security levels for DAA algorithms. In this section we will discuss the weaknesses and several of the papers that have aimed to improve on the initial API. We will then discuss the state

of the TPM 2.0 APIs for DAA as of the implementations considered in this thesis.

3.3.1 TPM 2.0: A Static Diffie-Hellman Oracle

After the initial publication of the TPM 2.0 APIs for DAA, researchers from the Chinese Academy of Sciences in Beijing noticed that the API introduced a weakness [49]. The TPM signature primitive, `TPM2_Sign`, could be used as a static Diffie-Hellman (SDH) oracle. Suppose G is a cyclic group of prime order n and x is a value in $\mathbb{Z}_n^*$. Given any $p \in G$ as input, a static Diffie-Hellman oracle on x outputs p^x. In this case the TPM 2.0 API produces a SDH oracle on the TPM secret key [49]. This research showed that the security level of a DAA implementation would be lowered as a result of this flaw [49]. The authors proposed a fix for this flaw in their paper.

Unfortunately, later research showed that the fix introduced a new problem! The modification to the API suggested by Xi et. al. introduces a subliminal channel [38]. This means the fix needs a fix! Introducing a nonce generated jointly by the host and the TPM solves this problem [38, 16]. These changes have been proposed to the TCG for inclusion in a new revision of the TPM 2.0 standard but have yet to be included in the new standard. As such, the implementations described in this thesis still suffer from the original SDH oracle problem. When the standard is updated and the changes propagate to the TPM libraries and software, the implementation can be updated with the new interface.

Chapter 4

Supporting Libraries & Implementation

In this chapter we will discuss the work required to implement two direct anonymous attestation schemes, EPID [13] and LASER [36]. We will first discuss the various sections of the TPM 2.0 specification and strategies to read the specification in order to understand how to utilize the features of the device. Next, we will discuss the software libraries utilized for implementing host and issuer computations as well as interacting with the TPM and how to work with them. Finally, we will discuss specifics regarding the implementation itself and issues that may arise when working with the TPM.

4.1 The TPM 2.0 Specification

The TPM 2.0 specification is over 1400 pages long and split into four parts [45]. In order to understand how to interact with the TPM it is necessary to read multiple parts of the standard. These parts are:

1. Part 1 Architecture: This section includes a description of the properties, functions, and methods of a TPM device. It also provides narrative explanations for the design decisions made.

2. Part 2 Structures: This section describes the constants, data types, structures, and unions used in the TPM interface and transferred to and from the device.

3. Part 3 Commands: This section contains a description of commands, parameters and response formats, and C code for the actions performed by the TPM. TPMs that follow the standard must provide identical results at the TPM interface as this C code.

4. Part 4 Supporting Routines: This section describes the algorithms and methods used internally by commands defined in the previous section.

It can be quite challenging to understand the standard by jumping directly to look at the commands you wish to work with in Part 3. Luckily, the current TPM editor and a few engineers who have worked with the TPM 2.0 standard produced a book [3] that includes an explanation of how to approach the standard in chapter 5. As of this writing an electronic

copy of the book can be found for free online. The advice of the three contributors differs slightly in terms of the best approach for beginning to read the standard [3].

4.1.1 Informal Advice on Reading the Specification

Starting by reading Part 1 to gain an understanding of the fundamental goals and limitations of the TPM may be the most productive. As Part 1 is primarily narrative, it does not provide detailed information on how to interact with the TPM. However, it explains the kinds of operations the TPM supports, the kinds of keys, authorization, and general functionality the TPM aims to achieve. In particular, it is helpful to read the section on 'Command/Response Structure' in Part 1 before reading Part 3 to gain an understanding of the markings and components of a command. Only after obtaining a basic understanding of the purposes of contexts, authorization sessions and handles should one progress to reading Part 3. If reading the specification in order to implement system-level software rather than using middleware like the Trusted Software Stack [29], it will also be necessary to read the discussion of canonicalization and the conversion of structures to byte streams to be sent to and from the TPM.

An engineer reading Part 3 of the specification is likely trying to identify which commands are needed to perform a particular operation. Once a candidate command is identified the command and response parameters can be analyzed to determine if the command does indeed perform the desired operation. Part of this analysis is to search for the structures

referenced in the command table in Part 2 of the specification. There may be multiple levels of structures before the results are clearly labelled, e.g. as group elements. As TPM commands are often designed to be generic, the reader can often only ascertain what the possible inputs and outputs of the command actually are by hopping through the first three Parts of the specification.

4.2 IBM Trusted Software Stack (TSS)

Much of the complexity in the TPM standard can be abstracted away by utilizing the Trusted Software Stack (TSS) [29]. The TSS is a software package that can be linked to applications to serve as a communication layer between the application and the TPM device. This functionality is extremely helpful and should be utilized by anyone writing applications that do not require lower-level access to read and write TPM structures or manipulate the byte stream. The TSS was used to implement the DAA schemes presented in this thesis.

4.2.1 Sending Commands to the TPM

The IBM TSS API is relatively straightforward. It consists of `TSS_Execute`, `TSS_Create`, and `TSS_Delete`. The API also offers customization through the `TSS_SetProperty` function. TSS contexts are opaque objects that keep track of parameters that influence the behavior of the TSS. Contexts are created with `TSS_Create` and deleted by `TSS_Delete`. The most interesting command in the API is `TSS_Execute`, which is used to pass input parameters

and a command designation to the TPM and to retrieve its output. `TSS_Execute` handles canonicalization and byte streams itself, the user need only understand the structures that are passed into and out of a TPM 2.0 command and the configuration options needed to make the TPM perform the desired operation.

By far the most challenging part of using the TSS is determining what configuration options are needed for a given command. Analysis of the examples provided with the library can make it much easier to implement common use cases but there is no alternative but to delve into the TPM 2.0 specification if a particular use case is not already implemented. `TSS_Execute` input requires a command parameters structure to be passed to execute a command; the specific fields of this structure are command-dependent. Similarly, the fields in the output structure are dependent on the specific command being executed *and* the configuration options that were passed as input. The best strategy to implement a new functionality is to read the many provided examples and determine where changes need to be made based on Parts 2 & 3 of the TPM specification [45].

4.2.2 Interfacing with the TPM Emulator

Along with the Trusted Software Stack, IBM has implemented a TPM 2.0 emulator that can be used to develop software for use with a TPM [28]. The emulator is helpful because it prevents early-development phase software run on a hardware TPM from locking you out of your device or deleting important information already stored on the physical TPM. It also

gives you more access to the internal state of the device than does a hardware TPM. The Trusted Software Stack makes it relatively easy to interface with the software TPM; simply follow the instructions that come with both software packages and change port designations if necessary. Once an application works on the TPM emulator, it can be easily ported to work with a hardware TPM instead.

4.2.3 Interfacing with the Hardware TPM

For this thesis we will consider implementations and testing on Linux systems only. Windows systems are also supported by the TSS but require a slightly different approach. In order to work with a hardware TPM directly on Linux, the TPM device driver must be installed (typically as a kernel module) and loaded in your kernel. It may be necessary to modify your BIOS settings to enable the security chip. The device will then be visible to the system as `/dev/tpm0`. To make the TSS direct commands to a hardware TPM instead of a TPM emulator, it is necessary to rebuild the TSS with flag `-DTPM_INTERFACE_TYPE_DEFAULT="dev"`. Then modify the properties of the TSS contexts in your TPM code as follows:
`TSS_SetProperty(tssContext, TPM_INTERFACE_TYPE, "dev");`. Given that the TPM is functional and your user has permissions to use it, and your TPM code properly handles authorization, this should be sufficient to port a working emulator implementation to the hardware TPM [29, 28].

```
type f
q 115792089237314936872688561244471742058375878355761205198700409522629664518163
r 115792089237314936872688561244471742058035595988840268584488757999429535617037
b 3
beta -2
alpha0 1
alpha1 1
```

Figure 4.1: BN P256 Curve Parameters for PBC

4.3 Pairing-based Cryptography (PBC) Library

Not all of the computations related to a DAA scheme happen on the TPM. For those operations that must be performed by the host and issuer the Pairing-based Cryptography (PBC) library can be used [39]. This library implements an interface and the underlying operations for elliptic curve addition, multiplication, and exponentiation and pairings. It is open-source, easily available, and supports multiple architectures.

4.3.1 Parameter Selection

As the TPM has only a limited set of elliptic curves that are supported it is necessary to parameterize the PBC library calls to use a supported curve. The curve used by the TPM that supports pairing operations is BN P256 [5, 44]. This curve is not supported directly by the PBC library, but it can be used by providing a parameter file for a 'Type f' curve. The parameter file used in this implementation is shown in Figure 4.1.

4.4 Using PBC and TSS

This chapter has discussed, up to this point, the libraries and documentation needed to prepare to implement DAA on TPM 2.0. The rest of Chapter 4 will discuss specific implementation details. The DAA schemes implemented in this thesis are EPID and LASER. This section will describe what features of the PBC and TSS libraries are needed for the implementations.

4.4.1 TPM 2.0 Commands Needed

In the discussion of related works in Chapter 3, the TPM 2.0 API for DAA was discussed. The TPM 2.0 standard includes numerous commands built into the system, only a few of these are actually required to implement a direct anonymous attestation scheme. Once again now we consider the commands needed to implement a DAA algorithm. The commands used in this implementation were `TPM2_CreatePrimary`, `TPM2_Commit`, `TPM2_Sign`, `TPM2_Hash`, `TPM2_GetRandom`, `TPM2_StartAuthSession`, and `TPM2_FlushContext`.

Functionality Provided

1. `TPM2_StartAuthSession`: This command is used to start an authorization session with the TPM. The functionality available to the user depends on what type of authorization session is in use.

2. `TPM2_CreatePrimary`: This command is used to create a key pair on the TPM. This

key is stored in the TPM memory if produced by `TPM2_CreatePrimary`. Alternatively, an encrypted key can be generated on the TPM and stored on disk using `TPM2_Create` and loaded into TPM memory when it is needed.

3. `TPM2_Commit`: This command is the first half of a signature operation. It produces up to three points on the elliptic curve and outputs these while also storing a random exponent used to produce two of the points as a commit record. The third point is produced from a generated basepoint raised to the TPM secret exponent.

4. `TPM2_Hash`: This command is used to generate a digest from an input buffer. It also produces proof that the TPM generated the digest itself in some cases.

5. `TPM2_Sign`: This command retrieves the record associated with a given `TPM2_Commit` and finishes the signature. It signs a digest produced by the TPM using `TPM2_Hash`.

6. `TPM2_GetRandom`: This command is used to generate nonces on the TPM. Once proposed changes to the specification are accepted this command may not be required.

7. `TPM2_FlushContext`: This command is used to flush objects such as keys or authorization session handles from the TPM memory.

Both `TPM2_StartAuthSession` and `TPM2_CreatePrimary` are only used once per implementation. `TPM2_Commit`, `TPM2_GetRandom`, `TPM2_Hash`, and `TPM2_Sign` are used repeatedly in both schemes. `TPM2_FlushContext` is used twice at the end of execution to clear the TPM memory allocated for the primary key and the authorization session.

```
primaryHandle = primaryHandle;
inSensitive.sensitive.data.t.size = 0;
inSensitive.sensitive.userAuth.t.size = 0;
inPublic.publicArea.type = TPM_ALG_ECC;
inPublic.publicArea.nameAlg = halg;
inPublic.publicArea.objectAttributes.val = 0;
inPublic.publicArea.objectAttributes.val |= TPMA_OBJECT_FIXEDTPM;
inPublic.publicArea.objectAttributes.val |= TPMA_OBJECT_FIXEDPARENT;
inPublic.publicArea.objectAttributes.val |= TPMA_OBJECT_SENSITIVEDATAORIGIN;
inPublic.publicArea.objectAttributes.val |= TPMA_OBJECT_USERWITHAUTH;
inPublic.publicArea.objectAttributes.val &= ~TPMA_OBJECT_ADMINWITHPOLICY;
inPublic.publicArea.objectAttributes.val |= TPMA_OBJECT_NODA;
inPublic.publicArea.objectAttributes.val &= ~TPMA_OBJECT_RESTRICTED;
inPublic.publicArea.objectAttributes.val &= ~TPMA_OBJECT_DECRYPT;
inPublic.publicArea.objectAttributes.val |= TPMA_OBJECT_SIGN;
inPublic.publicArea.authPolicy.t.size = 0; /* empty policy */
inPublic.publicArea.parameters.eccDetail.symmetric.algorithm = TPM_ALG_NULL;
inPublic.publicArea.parameters.eccDetail.scheme.scheme = TPM_ALG_ECDAA;
inPublic.publicArea.parameters.eccDetail.curveID = TPM_ECC_BN_P256;
inPublic.publicArea.parameters.eccDetail.kdf.scheme = TPM_ALG_NULL;
inPublic.publicArea.parameters.eccDetail.kdf.details.mgf1.hashAlg = halg;
inPublic.publicArea.parameters.eccDetail.scheme.details.ecdaa.hashAlg = halg;
inPublic.publicArea.unique.ecc.x.t.size = 0;
inPublic.publicArea.unique.ecc.y.t.size = 0;
outsideInfo.t.size = 0;
creationPCR.count = 0;
```

Figure 4.2: `TPM2_CreatePrimary` Command Parameters for Unrestricted ECDAA Key

Command Parameters

Each of the commands listed above has multiple different modes of operation. This is because the TPM hardware provides general functionality for each individual command and software options control the specific operation of the device [45]. Figure 4.2 shows the command options input to the TPM for the `TPM2_CreatePrimary` command with NULL

authorization to generate an unrestricted DAA key pair using the elliptic curve BN P256. Similar configuration blocks are used for each of the commands run on the TPM. In order to know what the configuration options are used for it is necessary to refer to the specification [45]. Part 3 will list which structures are input and output. Part 2 will list what fields are present in those structures. Part 1 will explain what the authorization and objectAttributes flags should be set to for a particular command.

4.4.2 PBC API

The last few pages may have made it seem that the majority of the code used in a DAA implementation is TPM code. This is far from the truth! The host, issuer, and verifier must also perform a number of computations beyond the host's interactions with the TPM. The PBC API makes implementing these computations much simpler than it otherwise would be. Most of the necessary operations are arithmetic over elliptic curve groups in addition to an occasional pairing or application of a hash function.

To begin working with the PBC library it is first necessary to initialize a pairing [39]. Figure 4.1 shown earlier is the parameter file for the BN P256 curve that is used by the TPM. A reference to a pairing is maintained by the `pairing_t` structure. The PBC library implements its arithmetic operations over a structure called `element_t`. These `element_t` structures are initialized as members of either G_1, G_2, G_T, or $\mathbb{Z}_r$ depending which group associated with the pairing they come from. For example, to initialize a G_1 element you would call

`element_init_G1(element, pairing)`. Arithmetic on elements is then handled with functions such as `element_add`, `element_mul`, `element_invert`, or `element_pow_zn`. Pairing operations can be applied using the aptly named function `element_pairing`. As the PBC library has excellent documentation on its website, we will consider this sufficient discussion of its use. Please read the manual [39].

4.5 General Implementation Notes

In general, most of the lines of code in each of the implemented schemes are `element_t` initialization, arithmetic, or clearing. The hash function used is SHA-256 and we utilize the implementation from OpenSSL [42]. Nonces are either generated by calls into `/dev/urandom` or by computing them with the TPM command `TPM2_GetRandom`.

4.5.1 Proving Knowledge with the TPM

The most common use of the TPM in DAA applications is generation of proofs of knowledge. Repeatedly in the next few sections discussing the implementation of EPID and LASER we will note that the TPM is used to prove knowledge of some secret value. In Chapter 2 we discussed the use of a hash function to convert an interactive zero-knowledge proof of knowledge into a signature via the Fiat-Shamir heuristic [27]. This is what the TPM does in this implementation. The signatures are produced with the command sequence `TPM2_Commit`, `TPM2_GetRandom`, `TPM2_Hash`, `TPM2_Sign`. An example of this process is shown in Figure 4.3.

`TPM2_Commit` was included in the TPM interface to make it possible to generate more diverse sets of signatures without needing several new commands. The commitment specifies all of the information that may need to be proven in advance; depending on the proof sign then reveals only the information that must be revealed later in the `TPM2_Sign` operation [30]. In specific, one of the elliptic curve points is exponentiated by the TPM secret key (let's call it f) for proofs which must include a relationship such as $K = B^f$ or $K \neq B^f$.

Following the generation of advance information in `TPM2_Commit`, the host must perform a few additional operations. It must generate random numbers and finish the joint computation of the commitment values that was started by the TPM. This can be thought of as randomization of the values generated by the TPM. Next, it computes a hash of the commitment values and the elliptic curve points that were input to the TPM. This digest is then passed to the TPM.

In future implementations after proposed changes are accepted into the TPM standard [38, 16], there will be a nonce generated by `TPM2_Commit` that will be used again in `TPM2_Sign`. Although this does not yet happen, we simulate this behavior by generating a nonce on the TPM instead using the `TPM2_GetRandom` command. This nonce is then hashed along with the digest produced by the host using `TPM2_Hash`. This hash operation is used to make the proof non-interactive. The digest produced by the TPM is then signed using `TPM2_Sign` and sent back to the host. The signature includes both the final digest T and a value of the form $r + T^f$ where r is the random number generated during `TPM2_Commit` and f is again the TPM secret. The host then finishes the zero-knowledge proof computation.

```
signProceedRevocationTPMCommit(bsn,
    bsni_pre, bsni_y, Ki,
    tid1, tid2, cntr);

element_random(gamma);
element_invert(inv_nymi, revEntry->nymi);
element_mul(Pi_i->Ci, Ki, inv_nymi);
element_pow_zn(Pi_i->Ci, Pi_i->Ci, gamma);

element_set1(one_G1);
if (!element_cmp(Pi_i->Ci, one_G1))
    printf("Ci == 1, verification fails\n");

element_random(rib);
element_invert(inv_nym, nym);
element_pow2_zn(ti1, tid1, gamma, inv_nymi, rib);
element_pow2_zn(ti2, tid2, gamma, inv_nym, rib);

Hash2(cd, Pi_i->Ci, revEntry->bsni, signRecord->bsn, revEntry->nymi, nym,
      ti1, ti2, NULL, NULL, NULL, 6);

/* This command computes a nonce with TPM2_GetRandom,
 * hashes the input and the nonce with TPM2_Hash,
 * then signs the digest with TPM2_Sign
*/
signProceedRevocationTPMSign(*cntr, cd, 32, Pi_i->c,
    siad, &Pi_i->ni);

element_mul(Pi_i->sia, gamma, siad);
element_mul(Pi_i->sib, Pi_i->c, gamma);
element_add(Pi_i->sib, rib, Pi_i->sib);
```

Figure 4.3: Example of Joint Host/TPM Computation of a Zero-Knowledge Proof

```
/* input the point h1 and configure options for TPM2_Commit */
commitIn.signHandle = pkey_handle;
commitIn.P1.size = 32 + 32 + 2 + 2;
commitIn.P1.point.x.t.size = 32;
commitIn.P1.point.y.t.size = 32;
memcpy(commitIn.P1.point.x.t.buffer, P1_x, 32);
memcpy(commitIn.P1.point.y.t.buffer, P1_y, 32);
commitIn.s2.t.size = 32;
memcpy(commitIn.s2.t.buffer, s2, 32);
commitIn.y2.t.size = 32;
memcpy(commitIn.y2.t.buffer, y2, 32);

/* call TPM2_Commit */
TSS_Execute(tssContext,
    (RESPONSE_PARAMETERS *)&commitOut,
    (COMMAND_PARAMETERS *)&commitIn,
    NULL,
    TPM_CC_Commit,
    auth_handle, NULL, 1,
    TPM_RH_NULL, NULL, 0);

/* Output */
memcpy(K_x, commitOut.K.point.x.t.buffer, 32);
memcpy(K_y, commitOut.K.point.y.t.buffer, 32);
memcpy(L_x, commitOut.L.point.x.t.buffer, 32);
memcpy(L_y, commitOut.L.point.y.t.buffer, 32);
memcpy(E_x, commitOut.E.point.x.t.buffer, 32);
memcpy(E_y, commitOut.E.point.y.t.buffer, 32);
*cntr = commitOut.counter;
```

Figure 4.4: TSS Code for Executing `TPM2_Commit`

```
getRandomNonce(nonce); /* Calls TPM2_GetRandom */
nonce_len = sizeof(uint32_t);
sizeInBytes = TSS_GetDigestSize(halg);
messageLength = nonce_len + hashbuffer_len;
message = malloc(messageLength);
memcpy(message, nonce, nonce_len);
memcpy(message + nonce_len, hashbuffer, hashbuffer_len);

hashIn.hierarchy = TPM_RH_NULL;
hashIn.data.t.size = messageLength;
hashIn.hashAlg = halg;
memcpy(hashIn.data.t.buffer, message, messageLength);

TSS_Execute(tssContext,
    (RESPONSE_PARAMETERS *)&hashOut,
    (COMMAND_PARAMETERS *)&hashIn,
    NULL, TPM_CC_Hash,
    TPM_RH_NULL, NULL, 0);

signIn.keyHandle = pkey_handle;
signIn.inScheme.scheme = TPM_ALG_ECDAA;
signIn.inScheme.details.ecdaa.count = cntr;
signIn.inScheme.details.ecdaa.hashAlg = halg;
signIn.digest.t.size = sizeInBytes;
memcpy(&signIn.digest.t.buffer,
(uint8_t *) hashOut.outHash.t.buffer, sizeInBytes);
signIn.validation.tag = TPM_ST_HASHCHECK;
signIn.validation.hierarchy = TPM_RH_NULL;
signIn.validation.digest.t.size = 0;

TSS_Execute(tssContext,
    (RESPONSE_PARAMETERS *)&signOut,
    (COMMAND_PARAMETERS *)&signIn,
    NULL, TPM_CC_Sign,
    auth_handle, NULL, 1,
    TPM_RH_NULL, NULL, 0);

/* Copy out signature components */
memcpy(r, signOut.signature.signature.ecdaa.signatureR.t.buffer, 32);
memcpy(s, signOut.signature.signature.ecdaa.signatureS.t.buffer, 32);
```

Figure 4.5: TSS Code for Executing `TPM2_Hash` and `TPM2_Sign`

The code on the previous few pages demonstrates an example zero-knowledge proof production. Figure 4.3 shows an abbreviated segment of code in which the host and TPM collaboratively produce a proof that the platform did not produce a revoked token. The code shown is only about 30% of the actual lines of code required to be executed by the host to implement the functionality; missing is memory allocation and deallocation, memory transfers, and error-handling code. The TPM code executed is shown in Figures 4.4 and 4.5. In both of those figures all error-handling is also removed to make the code more presentable. The function call `signProceedRevocationTPMCommit` in 4.3 and the function call to `signProceedRevocationTPMSign` trigger execution of code similar to Figures 4.4 and 4.5, respectively. Every proof of knowledge generated by the host and TPM in both the EPID and LASER implementations are based on the same core functionality presented in this example.

4.5.2 Validating Proofs of Knowledge

When the host and TPM produce a proof of knowledge in this manner they then send the values needed to produce the digest signed by the TPM and the signed digest itself to the verifier. The verifier then reproduces the digest and hashes it alongside the nonce generated by the TPM to validate the proof of knowledge. Sample code is shown below in Figure 4.6 that illustrates this process. This code is stripped of most error checking and memory allocations and deallocations to fit on the page. This code is used in the EPID implementation to validate proof that the platform did not produce a revoked signature.

```
/* Pi_i contains the signature components generated by the platform */
element_set1(one_G1);

if (!element_cmp(Pi_i->Ci, one_G1)) {
    printf("Verify not passed! :/ \n");
    return -1;
}

element_set1(one);
element_invert(inv_nymi, revEntry->nymi);
element_invert(inv_nym, nym);
element_pow2_zn(ti1h, revEntry->bsni, Pi_i->sia, inv_nymi, Pi_i->sib);
element_pow2_zn(ti2h, signRecord->bsn, Pi_i->sia, inv_nym, Pi_i->sib);
element_neg(tmpr, Pi_i->c);
element_pow2_zn(ti1h, ti1h, one, Pi_i->Ci, tmpr);

/* Reproduce digest sent to TPM to be signed */
Hash2(cp, Pi_i->Ci, revEntry->bsni, signRecord->bsn, revEntry->nymi, nym,
      ti1h, ti2h, 0, 0, 0, 6);

/* Reproduce value produced by TPM by hashing digest and nonce */
memcpy(ibuf, &Pi_i->ni, sizeof(uint32_t));
element_to_bytes(ibuf + sizeof(uint32_t), cp);
SHA256(ibuf, sizeof(uint32_t) + lz, obuf);

/* Validate hashes are equal */
element_from_bytes(cq, obuf);
if (element_cmp(cq, Pi_i->c)) {
    printf("\n Verify hashes neq! :( \n");
    return -1;
}
return 0;
```

Figure 4.6: Verifier Code for Validation of a Proof Of Knowledge

4.6 Implementing EPID

As discussed previously, EPID has five algorithms; `Setup`, `Join`, `Sign`, `Verify`, and `Revoke` [13]. Of these commands only `Join` and `Sign` require TPM computations.

1. `Setup` is run by just the issuer and is used to generate the group public key and the issuer secret key. This command is implemented entirely with the PBC library which is used for generating random elements, generating an exponent to serve as the secret key, and then exponentiation of a base by the random exponent to produce public values.

2. `Join` is run as a collaboration between the issuer and the platform comprised of the host and the TPM. First, if a key is not already generated, the TPM must generate a key pair and give the public part to the host. This is done via a call to `TPM2_CreatePrimary` in this implementation. The platform sends a join request to the issuer that includes a proof of knowledge of the TPM secret generated using calls to `TPM2_Commit`, `TPM2_GetRandom`, `TPM2_Hash` and `TPM2_Sign`. The issuer then validates the produced proof of knowledge. Next, the issuer starts a BBS+ signature and sends the generated values to the host. These values are generated using PBC. The host completes the BBS+ signature and validates the pairing holds using PBC operations.

3. `Sign` requires the platform to prove not only that is has a valid membership but also that it was not revoked. Both types of proof require some computation by the host platform in PBC and the same set of four TPM operations (`TPM2_Commit`, `TPM2_GetRandom`,

`TPM2_Hash`, `TPM2_Sign`) used to produce a proof of knowledge in `Join`. Proving membership requires the host to manipulate the BBS+ signature it received during `Join` and then run the sequence of TPM commands to prove it knows the secret associated with that BBS+ signature. Proving that the platform key has not been revoked requires the host to prove iteratively for each entry in the signature revocation list that its own secret could not have been used to produce the signature. Each signature is associated with a basename and comes accompanied with that basename raised to the TPM secret. This pair is used in the proof, if the TPM secrets do not match, the signature could not have been produced by this platform. Every proof generated in this manner requires each of the three calls to the TPM referenced above.

4. `Verify` and `Revoke` are both implemented entirely with the PBC library. Verification just requires that the verifier validate the proof of membership and each proof of non-revocation. `Revoke` adds the pair of values (basename and basename raised to TPM secret) from a signature to the revocation list in the case of signature-based revocation and adds the TPM secret key to the private key revocation list if it is exposed.

4.7 Implementing LASER

As many of the implemented functions of LASER correspond strongly to the same functions in EPID, we will spend less time considering the details of the implementation of LASER than EPID. In this section we will discuss what operations are required in each LASER

function and point out where the implementation diverges from EPID. LASER utilizes TPM commands in `Join`, `GetSignCre`, and `Sign`. The implementations of `Setup`, `Verify`, and `Revoke` are implemented purely with operations in the PBC library.

1. `Setup` is implemented using solely the PBC library. Similarly to EPID, in LASER this command essentially just generates the issuer secret key and group public key using a few calls to the PBC library.

2. `Join` generates a TPM secret key using `TPM2_CreatePrimary` and then proves its knowledge of this key using `TPM2_Commit`, `TPM2_GetRandom`, `TPM2_Hash`, and `TPM2_Sign`. This proof is sent to the issuer to be validated. The issuer responds with a membership credential, which is generated using PBC.

3. `GetSignCre` is a command that is not present in EPID. This command is used to obtain a signing credential that can be used for subsequent signatures. The command is implemented in a manner similar to the EPID sign operation; it requires the platform to prove its membership and prove that its TPM secret could not have been used to produce any revoked signatures. Both of these types of proofs are generated using the familiar `TPM2_Commit`, `TPM2_GetRandom`, `TPM2_Hash`, `TPM2_Sign` procedure. Once the proofs are validated by the issuer, the issuer and host jointly generate some number of alias tokens; generation starts at the issuer and a value is filled in at the host. The collection of alias tokens represents a given signing credential. All of the alias tokens are generated using PBC library functions.

4. `Sign` is a much simpler operation in LASER as its goal is to move the complexity offline to the process of obtaining credentials. This function accepts an alias token as an input and computes a proof of knowledge of the secret associated with that alias token using the four necessary commands. This proof demonstrates both demonstrates membership and that the platform is not revoked (or, at least, that the platform was not revoked when it obtained the signing credential!)

5. `Verify` is also simple; the verifier simply validates the single proof produced by the platform. All of these operations are performed with the PBC library. A SHA-256 hash is produced using OpenSSL.

6. `Revoke` first validates that the signature being revoked is actually a valid signature. These computations are performed with PBC. Then it simply adds entries from the alias token associated with a signature to the signature-based revocation list.

4.8 Issues

As discussed in Chapter 3, the TPM acts as a Static Diffie-Hellman oracle with respect to its secret key when computing DAA signatures [49]. There have been a number of proposed fixes to this problem but they have not yet been accepted into the standard, much less propagated to the TPM firmware [38, 16, 49, 23]. As a result, the schemes implemented in this thesis still use the TPM as a Static DH oracle. This reduces the overall security of the implementations. Once the TPM specification has been updated and the changes propagate

to the device drivers, the scheme can be easily modified to use the new interface. Most of the changes that have been proposed involve slight modifications to the `TPM2_Commit` and `TPM2_Sign` commands and are related to nonce generation [38, 16]. The nonces generated by these commands would serve to replace calls to `TPM2_GetRandom` by ensuring that the value produced by `Get_Random` could not have been modified prior to being sent back to the TPM.

Chapter 5

Evaluation of DAA Implementations

In this chapter we consider the relative performance of EPID and LASER. We evaluate both implementations to determine execution time as the size of the revocation list grows. We perform experiments to evaluate whether or not performance benefits can be obtained using the online/offline tradeoff or adaptable unlinkability in LASER relative to EPID for applications which require low online latency and allow for precomputation.

5.1 Experiments

The goal is to determine whether or not the online/offline tradeoff or adaptable unlinkability provide a performance benefit for LASER relative to EPID. Each of these questions is answered with a separate experiment.

5.1.1 Testbench

The implementations were tested on a Lenovo T460 laptop running Fedora Linux 25 with a 2.6 GHz Intel i7 6600U CPU with an onboard TPM running 2.0 firmware.

5.1.2 Testing the Online/Offline Tradeoff

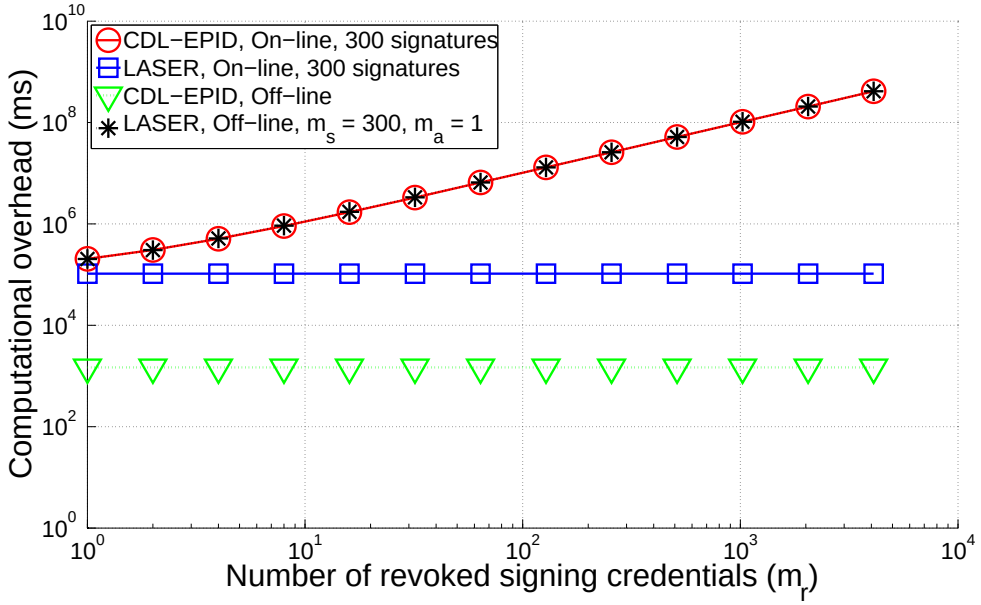

Figure 5.1: Online-Offline Performance of EPID and LASER

The experiment compared EPID to LASER by computing 300 signatures with both schemes. In order to make the comparison as fair as possible a fresh signing credential was used for each signature in LASER. This means each signature produced by LASER was fully unlinkable. The results are shown in Figure 5.1. From the figure it is clear that the online cost of LASER is much less than EPID for large revocation lists. However, the overall cost of LASER relative to EPID is slightly greater. This is due to the cost of obtaining multiple types of credentials.

5.1.3 Testing Adaptive Unlinkability

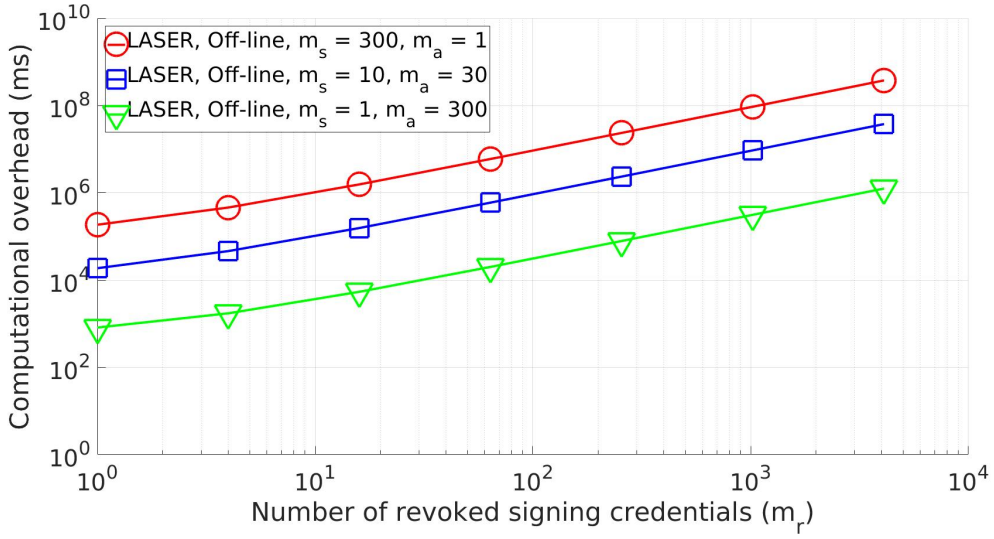

Figure 5.2: Performance Benefits of Adaptable Unlinkability

In order to determine the performance benefit of adaptable unlinkability for LASER, we designed another experiment that compared the total time taken to sign 300 messages using different numbers of signing credentials and alias tokens. The first case was the same as the original experiment; fully unlinkable LASER with 300 signing credentials (m_s) and a single alias token (m_a) per signing credential. The second case was to utilize 10 signing credentials and sign thirty messages under each credential using separate alias tokens. Finally, another test sequence used a single signing credential and signed each message under a separate alias token. The results of these experiments are shown in Figure 5.2. As the online cost of signing is consistent regardless of the type of credential used, the offline cost is graphed alone. The performance benefit of using adaptable unlinkability is clear. Utilizing adaptable unlinkability provided a benefit of two orders of magnitude when signing the 300 messages.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

With the TPM 2.0 specification approaching stability and driver software proliferating across popular operating systems, few barriers remain to the implementation of direct anonymous attestation schemes. Software stacks such as the IBM TSS [29] and TPM emulators [28] make interfacing with the TPM, developing software for the TPM, and testing implementations much easier than previous system-level APIs. At this point, the major barrier for the proliferation of DAA in practical settings is an inefficient revocation procedure. LASER, GSPR [37], and other works have aimed to reduce the cost of revocation by utilizing an online/offline tradeoff or probabilistic revocation. These efforts have yielded practical benefit, but there is still substantial room for improvement.

This work has shown that when direct anonymous attestation schemes are implemented to utilize a hardware TPM (rather than purely in software) the computation cost and time required for the computations on the TPM can become prohibitive very quickly. Further, we have shown that relaxing the strict unlinkability guarantees of EPID and other DAA schemes can substantially reduce the overhead of revocation.

6.2 Future Work

This work represents an early implementation of direct anonymous attestation schemes on commodity hardware, but a major goal of privacy-preserving authentication stakeholders is to port algorithms such as DAA to low-power and mobile devices. Currently most mobile phones are not shipped with TPM hardware and many low-power devices also do not include the hardware. Even if the devices did contain TPMs, it does not appear from the analysis of the implemented schemes that DAA is computationally feasible for typical application scenarios on lightweight and low-power devices. Improvements to the existing algorithms and additional testing will be required to make DAA more attractive for real-world implementations.

Bibliography

[1] G. Allee. Intel epid sdk. `https://01.org/epid-sdk`. Accessed: April 3, 2017.

[2] ARM. Arm security technology building a secure system using trustzone technology. `http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf`. Accessed: April 1, 2017.

[3] W. Arthur and D. Challener. *A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security*. Apress, Berkely, CA, USA, 1st edition, 2015.

[4] F. Baldimtsi, J. Camenisch, M. Dubovitskaya, A. Lysyanskaya, L. Reyzin, K. Samelin, and S. Yakoubov. Accumulators with applications to anonymity-preserving revocation. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017.

[5] P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *International Workshop on Selected Areas in Cryptography*, pages 319–331, 2005.

[6] J. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology*, EUROCRYPT '93, pages 274–285, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[7] D. Boneh and X. Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, 2008.

[8] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. In *Advances in Cryptology - ASIACRYPT*, volume 2248, pages 514–532. Springer Berlin Heidelberg, 2001.

[9] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use pgp. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, WPES '04, pages 77–84, New York, NY, USA, 2004. ACM.

[10] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS)*, pages 132–145, 2004.

[11] E. Brickell, L. Chen, and J. Li. A new direct anonymous attestation scheme from bilinear maps. In *International Conference on Trusted Computing*, pages 166–178, 2008.

[12] E. Brickell, L. Chen, and J. Li. Simplified security notions of direct anonymous attestation and a concrete scheme from pairings. *International Journal of Information Security*, 8(5):315–330, 2009.

[13] E. Brickell and J. Li. Enhanced privacy ID from bilinear pairing for hardware authentication and attestation. In *IEEE Second International Conference on Social Computing (SocialCom)*, pages 768–775, 2010.

[14] E. Brickell and J. Li. A pairing-based DAA scheme further reducing TPM resources. In *International Conference on Trust and Trustworthy Computing (TRUST)*, pages 181–195, 2010.

[15] E. Brickell and J. Li. Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. *IEEE Transactions on Dependable and Secure Computing*, 9(3):345–360, May 2012.

[16] J. Camenisch, L. Chen, M. Drijvers, A. Lehmann, D. Novick, and R. Urian. One tpm to bind them all: Fixing tpm 2.0 for provably secure anonymous attestation. In *IEEE Symposium on Security and Privacy (IEEE S&P)*, 2017.

[17] J. Camenisch, M. Drijvers, and A. Lehmann. Anonymous attestation using the Strong Diffie Hellman assumption revisited. In *Proceedings of the 9th International Conference on Trust and Trustworthy Computing (TRUST)*, pages 1–20, 2016.

[18] J. Camenisch, M. Drijvers, and A. Lehmann. Universally composable direct anonymous attestation. In *IACR International Workshop on Public Key Cryptography*, pages 234–264, 2016.

[19] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Advances in Cryptology - CRYPTO*, pages 126–144. 2003.

[20] E. Cesena, H. Löhr, G. Ramunno, A.-R. Sadeghi, and D. Vernizzi. Anonymous authentication with tls and daa. In *Proceedings of the 3rd International Conference on Trust and Trustworthy Computing*, TRUST'10, pages 47–62, Berlin, Heidelberg, 2010. Springer-Verlag.

[21] D. Chaum and E. Van Heyst. Group signatures. In *Proceedings of the 10th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'91, pages 257–265, Berlin, Heidelberg, 1991. Springer-Verlag.

[22] L. Chen and J. Li. Revocation of direct anonymous attestation. In *International Conference on Trusted Systems*, pages 128–147, 2010.

[23] L. Chen and J. Li. Flexible and scalable digital signatures in TPM 2.0. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 37–48, 2013.

[24] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, pages 41–, Washington, DC, USA, 1995. IEEE Computer Society.

[25] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[26] R. Dutta, R. Barua, and P. Sarkar. Pairing-based cryptographic protocols: A survey, 2004.

[27] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO*, pages 186–194. Springer-Verlag, 1987.

[28] K. Goldman. IBM TPM 2.0 emulator. `http://ibmswtpm.sourceforge.net/ibmswtpm2.html`. Accessed: Nov 1, 2016.

[29] K. Goldman. IBM TPM 2.0 TSS. `http://ibmswtpm.sourceforge.net/ibmtss2.html`. Accessed: Nov 1, 2016.

[30] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, July 1991.

[31] J. A. Grant. The national strategy for trusted identities in cyberspace: Enhancing online choice, efficiency, security, and privacy through standards. *IEEE Internet Computing*, 15(6):80–84, Nov 2011.

[32] International Organization for Standardization. ISO/IEC 11889-1:2009. `http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html`. Accessed: Nov 1, 2016.

[33] International Organization for Standardization and International Electrotechnical Commission. ISO/IEC 20009 (under development).

[34] International Organization for Standardization and International Electrotechnical Commission. ISO/IEC 20008:2013, 2013.

[35] International Organization for Standardization and International Electrotechnical Commission. ISO/IEC 11889:2015, parts 1-4, 2015.

[36] V. Kumar, H. Li, P. Asokan, N. Luther, and J.-M. Park. Laser: Lightweight anonymous attestation scheme with efficient revocation. Preparing for submission.

[37] V. Kumar, H. Li, J. Park, K. Bian, and Y. Yang. Group signatures with probabilistic revocation: A computationally-scalable approach for providing privacy-preserving authentication. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 1334–1345, 2015.

[38] A. Lehmann. Direct anonymous attestation and tpm 2.0 getting provably secure crypto into the real-world. In *Real World Cryptography 2017*, New York, New York, Jan. 4-7 2017.

[39] B. Lynn. PBC: Pairing-based cryptography, 2017.

[40] M. Manulis, N. Fleischhacker, F. Gunther, F. Kiefer, and B. Poettering. Group signatures - authentication with privacy. Technical report, Group Signatures Study for BSI - German Federal Office for Information Security, 2012.

[41] Microsoft. Device health attestation. `https://technet.microsoft.com/en-us/library/mt750346.aspx`. Accessed: Nov 1, 2016.

[42] OpenSSL. Cryptography and SSL/TLS toolkit, 2017.

[43] R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '01, pages 552–565, London, UK, UK, 2001. Springer-Verlag.

[44] Trusted Computing Group. Algorithm registry: Revision 01.24. `https://www.trustedcomputinggroup.org/wp-content/uploads/TCG_Algorithm_Registry_Rev_1.24.pdf`. Accessed: April 14, 2017.

[45] Trusted Computing Group. TPM library specification. `http://www.trustedcomputinggroup.org/tpm-library-specification/`. Accessed: Nov 1, 2016.

[46] Trusted Computing Group. TPM main specification. `http://www.trustedcomputinggroup.org/tpm-main-specification/`. Accessed: Nov 1, 2016.

[47] Trusted Computing Platform Alliance. TPM main specification. `https://trustedcomputinggroup.org/tcpa-main-specification-version-1-1b/`. Accessed: April 1, 2017.

[48] L. Xi, D. Feng, Y. Qin, F. Wei, J. Shao, and B. Yang. Direct anonymous attestation in practice: Implementation and efficient revocation. In *Twelfth Annual International Conference on Privacy, Security and Trust (PST)*, pages 67–74, 2014.

[49] L. Xi, K. Yang, Z. Zhang, and D. Feng. DAA-related APIs in TPM 2.0 revisited. In *International Conference on Trust and Trustworthy Computing (TRUST)*, pages 1–18, 2014.

[50] B. Yang, D. Feng, and Y. Qin. A lightweight anonymous mobile shopping scheme based on DAA for trusted mobile platform. In *IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 9–17, 2014.

[51] B. Yang, K. Yang, Y. Qin, Z. Zhang, and D. Feng. *DAA-TZ: An Efficient DAA Scheme for Mobile Devices Using ARM TrustZone*, pages 209–227. Springer International Publishing, Cham, 2015.